

Document downloaded from:

<http://hdl.handle.net/10251/84125>

This paper must be cited as:

Defez Candel, E.; Sastre, J.; Ibáñez González, JJ.; Peinado Pinilla, J. (2016). Solving engineering models using hyperbolic matrix functions. *Applied Mathematical Modelling*. 40(4):2837-2844. doi:10.1016/j.apm.2015.09.050.



The final publication is available at

<https://doi.org/10.1016/j.apm.2015.09.050>

Copyright Elsevier

Additional Information

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/292881509>

Solving engineering models using hyperbolic matrix functions

Article in *Applied Mathematical Modelling* · February 2016

DOI: 10.1016/j.apm.2015.09.050

CITATIONS

0

READS

88

4 authors:



[Emilio Defez](#)

Universitat Politècnica de València

74 PUBLICATIONS 502 CITATIONS

[SEE PROFILE](#)



[Jorge Sastre Martínez](#)

Universitat Politècnica de València

63 PUBLICATIONS 366 CITATIONS

[SEE PROFILE](#)



[Jacinto Javier Ibáñez](#)

Universitat Politècnica de València

40 PUBLICATIONS 116 CITATIONS

[SEE PROFILE](#)



[Jesús Peinado](#)

Universitat Politècnica de València

23 PUBLICATIONS 51 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



High Performance Computing, Matrix algorithms. Lately, computing matrix exponential [View project](#)



Riccati Matrix Equations [View project](#)

All content following this page was uploaded by [Jesús Peinado](#) on 03 February 2016.

The user has requested enhancement of the downloaded file. All in-text references [underlined in blue](#) are added to the original document and are linked to publications on ResearchGate, letting you access and read them immediately.

Solving engineering models using hyperbolic matrix functions

Emilio Defez^{*}, Jorge Sastre[†], Javier Ibáñez[‡], Jesús Peinado[‡]

^{*} Instituto de Matemática Multidisciplinar.

[†] Instituto de Telecomunicaciones y Aplicaciones Multimedia.

[‡] Instituto de Instrumentación para Imagen Molecular.

Universitat Politècnica de València,

Camino de Vera s/n, 46022 Valencia, Spain.

edefez@imm.upv.es, jorsasma@iteam.upv.es, {jjibanez, jpeinado}@dsic.upv.es

Abstract

In this paper a method for computing hyperbolic matrix functions based on Hermite matrix polynomial expansions is outlined. Hermite series truncation together with Paterson-Stockmeyer method allow to compute the hyperbolic matrix cosine efficiently. A theoretical estimate for the optimal value of its parameters is obtained. An efficient and highly-accurate Hermite algorithm and a MATLAB implementation have been developed. The MATLAB implementation has been compared with the MATLAB function `funm` on matrices of different dimensions, obtaining lower execution time and higher accuracy in most cases. To do this we used an NVIDIA Tesla K20 GPGPU card, the CUDA environment and MATLAB. With this implementation we get much better performance for large scale problems.

Keywords. Hermite matrix polynomial, hyperbolic matrix functions, series expansion.

1 Introduction

Coupled partial differential problems are frequently found in many different fields of technology and science: elastic and inelastic contact problems of solids, biochemistry, magnetohydrodynamic flows, cardiology, diffusion problems, etc, see for example [1–4] and references therein. In particular, coupled hyperbolic systems appear in microwave heating processes [5] and optics [6] for instance.

In [7], an exact solution of coupled hyperbolic systems of the form

$$\left. \begin{aligned} u_{tt}(x, t) - Au_{xx}(x, t) &= 0, & 0 < x < 1, & t > 0 \\ u(0, t) + B_1 u_x(0, t) &= 0, & t > 0 \\ A_2 u(1, t) + B_2 u_x(1, t) &= 0, & t > 0 \\ u(x, 0) &= f(x), & 0 \leq x \leq 1, \\ u_t(x, 0) &= g(x), & 0 \leq x \leq 1, \end{aligned} \right\} \quad (1.1)$$

where A, B_1, A_2, B_2 are $r \times r$ complex matrices, and u, f, g are r -vector valued functions, was constructed in terms of a series which used hyperbolic cosine and sine of a matrix, respectively defined by

$$\cosh(Ay) = \frac{e^{Ay} + e^{-Ay}}{2}, \quad \sinh(Ay) = \frac{e^{Ay} - e^{-Ay}}{2}, \quad A \in \mathbb{C}^{r \times r}. \quad (1.2)$$

This work has been supported by Spanish Ministerio de Economía y Competitividad and European Regional Development Fund (ERDF) grant TIN2014-59294-P.

Functions of a matrix are used in many areas of science and arise in numerous applications in engineering, [8]. In particular, matrix exponential e^A and matrix functions sine and cosine have been those that have received the most attention, see [9–14] for example, not only for its computational difficulties but also for its importance in solving differential systems of first and second order.

For the numerical solution of problem (1.1), analytic-numerical approximations are most suitably obtained by truncation of the exact series solution given in [7]. Thus, we need to compute approximations of both matrix functions hyperbolic cosine and sine with good accuracy and efficiency. It is well known that this computation can be reduced to the computation of the cosine of a matrix, due to the identities

$$\cosh(A) = \cos(iA), \quad \sinh(A) = i \cos\left(A - \frac{i\pi}{2}I\right),$$

but this approach has the disadvantage, however, to require complex arithmetic even though the matrix A is real, which is usually the case in applications. Furthermore, it is possible to reduce the computation of $\sinh(A)$ to the computation of $\cosh(A)$ by the relation

$$\sinh(A) = -i \cosh\left(-A - \frac{\pi}{2}iI\right), \quad (1.3)$$

but obviously, formula (1.3) also requires complex arithmetic although matrix A is real, which contributes substantially to the computational overhead. Direct calculation through the exponential matrix using (1.2) is also costly. A method based on Hermite series of the hyperbolic matrix cosine was recently presented in the *17th European Conference on Mathematics for Industry 2012* (Lund, Sweden) and also published in [15]. In this paper a bound of the error was given, however, an algorithm for its calculation was not discussed. In the *Mathematical Modelling in Engineering & Human Behaviour 2013* Conference, (Valencia, Spain) [16] this same idea was applied to compute the sine and hyperbolic cosine of a matrix, finding different bounds, more accurate than those obtained in [15]. An algorithm based on the recurrence relation of three terms was developed, which allowed the simultaneous calculation of both functions. The proceedings of that conference were published as a book chapter [17] and the summaries were published in [16, 18].

In this work, we propose an algorithm to compute both matrix functions, $\sinh(A)$ and $\cosh(A)$, simultaneously, avoiding complex arithmetic whenever possible, following a similar approach to that used in [14] for computing approximations of the matrix cosine. The proposed method uses Hermite matrix polynomial expansions of both matrix functions in order to provide a very accurate and competitive method for computing them. For this purpose, firstly we have to determinate error bounds of the approximation error of both functions using a series of Hermite matrix polynomials, bounds different from and more accurate than the bounds presented in [15] and [17]. Then, we have developed an algorithm based on Paterson-Stockmeyer method [19], instead of the algorithm based on the recurrence relation of three terms, in which the corresponding scaling of the matrix and the order of the approximation have been determined from the bound expression. To analyze the accuracy of the proposed method, implementations have been developed in MATLAB. We have also performed numerous tests to compare this algorithm with other state-of-the-art algorithms (`funm` MATLAB function). Finally, high performance implementations have been developed on large dimension matrix problems by using GPGPU cards.

This work is organized as follows. Sections 2–3 summarizes previous results of Hermite matrix polynomials and includes a Hermite series expansion of the matrix hyperbolic cosine and sine with the respectively error bounds. An algorithm for the proposed method is given in section 4. Section 5 deals with several numerical tests in order to investigate the accuracy of the proposed method and the method behaviour for large scale problems. Both of them were done with MATLAB. When

evaluating the performance of the method for large scale problems, we also did a GPGPU (General Purpose Graphics Processing Unit) implementation. The idea is compare the CPU vs GPGPU implementations.

In this paper, $\lceil x \rceil$ rounds to the nearest integer greater than or equal to x . The matrices I_r and $\theta_{r \times r}$ in $\mathbb{C}^{r \times r}$ denote the matrix identity and the null matrix of order r , respectively. We will use subordinate matrix norms $\|A\|$, $A \in \mathbb{C}^{r \times r}$, and $\|A\|_1$ denotes the usual 1-norm. If $\mathcal{A}(k, n)$ is a matrix in $\mathbb{C}^{r \times r}$ for $n \geq 0, k \geq 0$, then [20]:

$$\sum_{n \geq 0} \sum_{k \geq 0} \mathcal{A}(k, n) = \sum_{n \geq 0} \sum_{k=0}^n \mathcal{A}(k, n-k). \quad (1.4)$$

2 Hermite matrix polynomial series expansions of hyperbolic matrix cosine. Error bound

The following properties of Hermite matrix polynomials which have been established in [13, 20, 21] will be used in this paper. From (3.4) of [21, p. 25] the n th Hermite matrix polynomial is defined by

$$H_n \left(x, \frac{1}{2} A^2 \right) = n! \sum_{k=0}^{\lfloor \frac{n}{2} \rfloor} \frac{(-1)^k (xA)^{n-2k}}{k!(n-2k)!}, \quad (2.1)$$

for an arbitrary matrix A in $\mathbb{C}^{r \times r}$. From [13], we obtain:

$$\cosh(Ay) = e^{\frac{1}{2}y^2} \sum_{n \geq 0} \frac{1}{\lambda^{2n} (2n)!} H_{2n} \left(y\lambda, \frac{1}{2} A^2 \right), \quad \lambda \in \mathbb{R}^+. \quad (2.2)$$

Denoting by $CH_N(\lambda, A^2)$ the N th partial sum of series (2.2) for $y = 1$, one gets the approximation

$$CH_N(\lambda, A^2) = e^{\frac{1}{2}} \sum_{n=0}^N \frac{1}{\lambda^{2n} (2n)!} H_{2n} \left(\lambda, \frac{1}{2} A^2 \right) \approx \cosh(A), \quad \lambda \in \mathbb{R}^+. \quad (2.3)$$

As in [22, pp. 1913], a bound for Hermite matrix polynomials $\|H_{2n}(x, \frac{1}{2}A^2)\|$ based on $\|A^2\|$ is obtained, see [23], using the Taylor series for the hyperbolic cosine $\cosh(y) = \sum_{n \geq 0} y^{2n}/(2n)!$. Taking norms in (2.1), one gets

$$\left\| H_{2n} \left(x, \frac{1}{2} A^2 \right) \right\| \leq (2n)! \sum_{k=0}^n \frac{\left(\|A^2\|^{\frac{1}{2}} \right)^{2(n-k)}}{k!(2(n-k))!} |x|^{2(n-k)}. \quad (2.4)$$

By using (1.4), it follows that

$$\begin{aligned} e \cosh \left(|x| \|A^2\|^{\frac{1}{2}} \right) &= \sum_{n \geq 0} \frac{\left(|x| \|A^2\|^{\frac{1}{2}} \right)^{2n}}{(2n)!} \sum_{k \geq 0} \frac{1}{k!} = \sum_{n \geq 0} \sum_{k \geq 0} \frac{\left(|x| \|A^2\|^{\frac{1}{2}} \right)^{2n}}{(2n)! k!} \\ &= \sum_{n \geq 0} \sum_{k=0}^n \frac{\left(\|A^2\|^{\frac{1}{2}} \right)^{2(n-k)}}{k!(2(n-k))!} |x|^{2(n-k)}, \end{aligned} \quad (2.5)$$

and as a consequence

$$\sum_{k=0}^n \frac{\left(\|A^2\|^{\frac{1}{2}} \right)^{2(n-k)}}{k!(2(n-k))!} |x|^{2(n-k)} \leq e \cosh \left(|x| \|A^2\|^{\frac{1}{2}} \right). \quad (2.6)$$

Multiplying by $(2n)!$ in (2.6) and using (2.4), we have the result:

$$\left\| H_{2n} \left(x, \frac{1}{2} A^2 \right) \right\| \leq (2n)! e \cosh \left(x \|A^2\|^{\frac{1}{2}} \right), \quad \forall x \in \mathbb{R}, \quad n \geq 0, \quad \forall A \in \mathbb{C}^{r \times r}. \quad (2.7)$$

Taking into account approximation (2.3) and bound (2.7), it follows that, for $\lambda \neq 0$:

$$\begin{aligned} \left\| \cosh(A) - CH_N(\lambda, A^2) \right\| &\leq e^{\frac{1}{\lambda^2}} \sum_{k \geq N+1} \frac{1}{\lambda^{2k} (2k)!} \left\| H_{2k} \left(\lambda, \frac{1}{2} A^2 \right) \right\| \\ &\leq e^{\frac{1}{\lambda^2} + 1} \cosh \left(\lambda \|A^2\|^{\frac{1}{2}} \right) \sum_{k \geq N+1} \frac{1}{\lambda^{2k}} \\ &= e^{\frac{1}{\lambda^2} + 1} \cosh \left(\lambda \|A^2\|^{\frac{1}{2}} \right) \left[\sum_{k \geq 0} \frac{1}{\lambda^{2k}} - \sum_{k=0}^N \frac{1}{\lambda^{2k}} \right]. \end{aligned} \quad (2.8)$$

Simplifying (2.8), we obtain finally the bound:

$$\left\| \cosh(A) - CH_N(\lambda, A^2) \right\| \leq \frac{e^{\frac{1}{\lambda^2} + 1} \cosh \left(\lambda \|A^2\|^{\frac{1}{2}} \right)}{(\lambda^2 - 1) \lambda^{2N}}. \quad (2.9)$$

3 Hermite matrix polynomial series expansions of hyperbolic matrix sine. Error bound

Analogously, one gets a similar expression for :

$$\sinh(Ay) = e^{\frac{1}{\lambda^2}} \sum_{n \geq 0} \frac{1}{\lambda^{2n+1} (2n+1)!} H_{2n+1} \left(y\lambda, \frac{1}{2} A^2 \right), \quad \lambda \in \mathbb{R}^+. \quad (3.1)$$

Denoting by $SH_N(A, \lambda)$ the N th partial sum of series (3.1) for $y = 1$, one gets

$$SH_N(\lambda, A) = e^{\frac{1}{\lambda^2}} \sum_{n=0}^N \frac{1}{\lambda^{2n+1} (2n+1)!} H_{2n+1} \left(\lambda, \frac{1}{2} A^2 \right) \approx \sinh(A), \quad A \in \mathbb{C}^{r \times r}. \quad (3.2)$$

Now we can obtain a bound for Hermite matrix polynomials $\|H_{2n+1}(x, \frac{1}{2} A^2)\|$ based on $\|A^2\|$, see [23], using the Taylor series for the hyperbolic sine $\sinh(y) = \sum_{n \geq 0} y^{2n+1} / (2n+1)!$. Taking norms in (2.1), one gets

$$\left\| H_{2n+1} \left(x, \frac{1}{2} A^2 \right) \right\| \leq (2n+1)! |x| \|A\| \sum_{k=0}^n \frac{\left(\|A^2\|^{\frac{1}{2}} \right)^{2(n-k)}}{k! (2(n-k)+1)!} |x|^{2(n-k)}. \quad (3.3)$$

By using (1.4), we obtain for $x \neq 0$:

$$\begin{aligned} \frac{e \sinh \left(|x| \|A^2\|^{\frac{1}{2}} \right)}{|x| \|A^2\|^{\frac{1}{2}}} &= \sum_{n \geq 0} \frac{\left(|x| \|A^2\|^{\frac{1}{2}} \right)^{2n}}{(2n+1)!} \sum_{k \geq 0} \frac{1}{k!} = \sum_{n \geq 0} \sum_{k \geq 0} \frac{\left(|x| \|A^2\|^{\frac{1}{2}} \right)^{2n}}{(2n+1)! k!} \\ &= \sum_{n \geq 0} \sum_{k=0}^n \frac{\left(\|A^2\|^{\frac{1}{2}} \right)^{2(n-k)}}{k! (2(n-k)+1)!} |x|^{2(n-k)}, \end{aligned} \quad (3.4)$$

and

$$\sum_{k=0}^n \frac{\left(\|A^2\|^{\frac{1}{2}}\right)^{2(n-k)}}{k!(2(n-k)+1)!} |x|^{2(n-k)} \leq \frac{e \sinh\left(|x| \|A^2\|^{\frac{1}{2}}\right)}{|x| \|A^2\|^{\frac{1}{2}}}. \quad (3.5)$$

Multiplying by $(2n+1)!|x|\|A\|$ in (3.5) and using (3.3), we have the result:

$$\left\| H_{2n+1}\left(x, \frac{1}{2}A^2\right) \right\| \leq (2n+1)! \frac{\|A\|}{\|A^2\|^{\frac{1}{2}}} e \sinh\left(|x| \|A^2\|^{\frac{1}{2}}\right), \quad \forall x \in \mathbb{R} \sim \{0\}, \quad n \geq 0, \quad \forall A \in \mathbb{C}^{r \times r}. \quad (3.6)$$

Since $\sinh(0) = 0$ and $H_{2n+1}\left(0, \frac{1}{2}A^2\right) = \theta_{r \times r}$, the bound (3.6) holds also when $x = 0$.

Taking into account (3.2) and the bound (3.6), it follows that, for $\lambda \neq 0$:

$$\begin{aligned} \left\| \sinh(A) - SH_N(\lambda, A^2) \right\| &\leq e^{\frac{1}{\lambda^2}} \sum_{k \geq N+1} \frac{1}{\lambda^{2k+1}(2k+1)!} \left\| H_{2k+1}\left(\lambda, \frac{1}{2}A^2\right) \right\| \\ &\leq e^{\frac{1}{\lambda^2}+1} \frac{\|A\|}{\|A^2\|^{\frac{1}{2}}} \sinh\left(|\lambda| \|A^2\|^{\frac{1}{2}}\right) \sum_{k \geq N+1} \frac{1}{\lambda^{2k+1}} \\ &= e^{\frac{1}{\lambda^2}+1} \frac{\|A\|}{\lambda \|A^2\|^{\frac{1}{2}}} \sinh\left(|\lambda| \|A^2\|^{\frac{1}{2}}\right) \left[\sum_{k \geq 0} \frac{1}{\lambda^{2k}} - \sum_{k=0}^N \frac{1}{\lambda^{2k}} \right]. \end{aligned} \quad (3.7)$$

Simplifying (3.7), we have the following bound:

$$\left\| \sinh(A) - SH_N(\lambda, A^2) \right\| \leq e^{\frac{1}{\lambda^2}+1} \frac{\|A\|}{\|A^2\|^{\frac{1}{2}}} \frac{\sinh\left(|\lambda| \|A^2\|^{\frac{1}{2}}\right)}{(\lambda^2 - 1)\lambda^{2N+1}}. \quad (3.8)$$

4 Hermite Algorithm

In this section we describe an algorithm based on Hermite series for computing the matrix hyperbolic cosine.

Algorithm 1 (herm) Given a matrix $A \in \mathbb{C}^{r \times r}$ and the order N of Hermite matrix approximation of the hyperbolic cosine function, this algorithm computes $B \cong \cosh(A)$.

- 1: Preprocessing of matrix A : \check{A} .
 - 2: Compute optimal values of λ and s .
 - 3: SCALING PHASE: $\check{A} = \check{A}/2^s$
 - 4: Compute $\check{B} = CH_N(\lambda, \check{A}^2)$, where CH_N is the Hermite approximation of the hyperbolic cosine function.
 - 5: **for** $i = 1 : s$ **do**
 - 6: $\check{B} = 2\check{B}^2 - I$
 - 7: **end for**
 - 8: Postprocessing of matrix \check{B} : B .
-

Preprocessing and postprocessing can be used to reduce the norm of matrix A . An available technique for that purpose is based on balancing [8, p. 299]. This technique attempts to balance the norms of the k th row and k th column, for each k , by means of a diagonal similarity transformation defined by a non singular matrix V . If

$$\check{A} = V^{-1}AV$$

is the obtained matrix in the preprocessing, then the postprocessing is as follows

$$B = V \cosh(\tilde{B})V^{-1}.$$

For the evaluation of $CH_N(\lambda, \tilde{A}^2)$ the Horner and Paterson-Stockmeyer's method can be applied [19,24], obtaining previously with MATLAB the symbolic expression of $CH_N(\lambda, \tilde{A}^2)$ as a polynomial of matrix \tilde{A}^2 with degree N and coefficients depending on λ , for each considered value of N . The formula $\cosh(2A) = 2 \cosh^2(A) - I_r$ is used to recover $\cosh(\tilde{A})$ from the matrix \tilde{B} obtained in Step 4.

The optimal values of N for computing $CH_N(\lambda, \tilde{A}^2)$ by means of the Paterson-Stockmeyer method belong to $S_N = \{1, 2, 4, 6, 9, 12, 16, 20, \dots\}$ [24, p. 6454]. If N_k is the order of Hermite approximation, where k is the position of N in S_N , then the number of matrix product evaluations in Algorithm 1 is $k + s$. For computing $\cosh(A)$ with IEEE double precision arithmetic, we take the error in (2.9) to be lower than or equal to the unit roundoff in double precision floating-point $u = 2^{-53}$, i.e,

$$\frac{e^{\frac{1}{\lambda^2} + 1} \cosh\left(\lambda \left\| \left(\tilde{A}/2^s\right)^2 \right\|^{\frac{1}{2}}\right)}{(\lambda^2 - 1)\lambda^{2N}} \leq u.$$

then

$$s \geq \frac{1}{2} \log_2(\|\tilde{A}^2\|) + \log_2\left(\frac{\lambda}{\operatorname{arc\,cosh}\left(\frac{u(\lambda^2 - 1)\lambda^{2N}}{e^{1+1/\lambda^2}}\right)}\right). \quad (4.1)$$

We choose the parameter $\lambda = \lambda_{\min}$, where λ_{\min} is the value such that the right-hand side of (4.1) reaches the minimum value. If we define

$$\Theta_N = \frac{1}{\lambda_{\min}} \operatorname{arc\,cosh}\left(\frac{u(\lambda_{\min}^2 - 1)\lambda_{\min}^{2N}}{e^{1+1/\lambda_{\min}^2}}\right) \quad (4.2)$$

then, using (4.1), it follows that

$$s = \left\lceil \log_2\left(\frac{\sqrt{\|\tilde{A}^2\|_1}}{\Theta_N}\right) \right\rceil. \quad (4.3)$$

We choose N in $\{1, 2, 4, 6, 9, 12, 16, 20\}$, because the corresponding values of λ_{\min} are negative for $N = 25, 30$. The values of λ_{\min} are listed in Table 1a. These values have been computed by using the symbolic functions `diff` and `solve` from the Symbolic Math Toolbox 5 of MATLAB. The values Θ_N of (4.2) are listed in Table 1b.

λ_{\min}		Θ_N	
$N=1$	28614.37029738510	$N=1$	$1.398832216450000 \cdot 10^{-4}$
$N=2$	1304.997141358828	$N=2$	$4.597769511080000 \cdot 10^{-3}$
$N=4$	110.4336118931741	$N=4$	$9.055511153551000 \cdot 10^{-2}$
$N=6$	38.32012920933002	$N=6$	$3.653432599794136 \cdot 10^{-1}$
$N=9$	17.32558067391524	$N=9$	1.154363749580479
$N=12$	11.29953801535487	$N=12$	2.300989971177028
$N=16$	8.081170359288837	$N=16$	4.207370311219608
$N=20$	6.566785645725286	$N=20$	6.395990872756508

(a) Optimal values of λ_{\min} .

(b) Values of Θ_N from (4.2).

Table 1: Tables of λ_{\min} and Θ_N .

There is no analogue of Algorithm 1 for computing the hyperbolic sine, because the corresponding formula $\sinh(2A) = 2 \sinh(A) \cosh(A)$ requires to compute hyperbolic cosine. However,

a similar algorithm to Algorithm 12.8 from [8, p. 298], which computes the sine and cosine of a matrix, can be developed to compute both the hyperbolic sine and hyperbolic cosine.

5 Numerical tests

5.1 MATLAB implementation

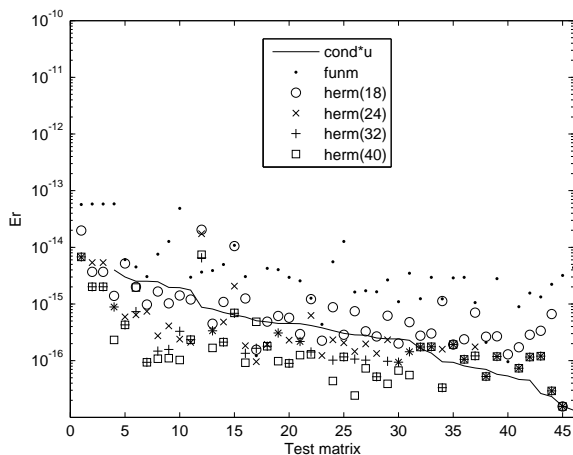
In this section a MATLAB implementation of Algorithm 1 has been compared with MATLAB function `funm`. We used an Apple Macintosh iMac (mid 2011) with a quadcore i5-2400S 2.5 Ghz processor and 12Gb of RAM. All the tests were carried out using MATLAB R2012a and OS X 10.6.8.

5.1.1 First Case Study

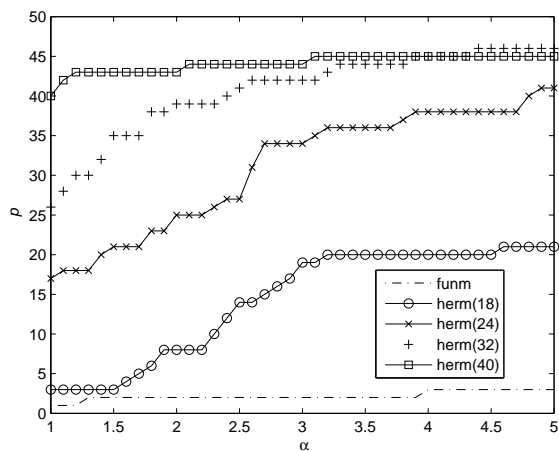
In the first case study we have used forty nine matrices from the Matrix Computation Toolbox [25] of dimension equal to 8. The “*exact*” value of $\cosh(A)$ has been calculated by using [32/32] diagonal Padé method with scaling and squaring with 64-digit precision in an iterative way: different increasing scalings starting from that provided in [26] for `expm` were used, until the norm of the relative difference between the approximations converted to IEEE double precision arithmetic was zero. The [32/32] diagonal Padé approximation was evaluated with a matrix power aggregation similar to that proposed in [26, p. 1183].

Figure 1a shows the normwise relative errors. This figure shows the relative errors of `funm` and `herm(m)`, where $m = 2N$ is the order of Hermite approximation, sorted in decreasing order, and a solid line that represents the unit roundoff multiplied by the relative condition number of the hyperbolic matrix cosine at X [8, p. 56]. For a method to perform in a backward and forward stable manner, its error should lie not far above this line on the graph [26, p. 1188]. Figure 1a shows that all functions perform in a numerically stable way on tests.

Performance profiles [27] are presented in Figure 1b. This figure shows the performances of the compared functions, where α varies between 1 and 5 in steps equal to 0.1, and p is the probability that the considered function has a relative error lower or equal than α -times the smallest error over all the methods. As shown in this figure, `herm` with maximum order $m = 40$ is the most accurate function, and it was achieved with very similar cost to `herm` with maximum orders $m = 18, 24, 32$ or 40 , i.e. $N = 9, 12, 16$ or 20 . Hence, we consider $m = 40$ as the best choice of maximum order for `herm`.



(a) Normwise relative errors



(b) Performance profile

Figure 1: Comparative for matrices of the Toolbox package.

Table 2: Table comparing the number of Gflops and time (in seconds) in CPU vs GPU.

Size	Speedup	GflopsCPU	GflopsGPU	T.CPU	T.GPU
128	0.97	24.14	23.53	0.02781	0.002867
256	2.13	43.18	92.33	0.09168	0.004304
512	4.81	50.74	244.58	0.05063	0.010526
1024	7.93	78.25	621.20	0.33054	0.041680
2048	8.81	91.92	810.18	1.83277	0.208033
4096	10.26	90.41	928.48	7.36381	0.717720
8192	10.28	96.09	987.88	14.1168	1.373200

5.1.2 Second Case Study

In the second case study we have used one hundred diagonalizable matrices of dimension equal to 512. These matrices have been generated as $A = QDQ$, where $Q = H/\sqrt{512}$, with H a Hadamard matrix of dimension 512. Diagonal matrices D have been randomly generated, with 2-norms varying between 1 and 100. The hyperbolic cosine of A has been computed as $\cosh(A) = Q \cosh(D)Q$, using 64 digits of precision.

For all matrices, the relative error of the Hermite implementation was lower than the relative error of `funm` function. The total time for all the 100 executions for our implementation is 5.77 seconds and for `funm` 21.32 seconds.

5.2 GPGPU implementation

Sometimes it is interesting to get the solution for large scale problems. In this case it is interesting to use techniques to improve the speed of the resolution of the problem as much as possible. One of these techniques consists of using one or several GPGPU (General Purpose Graphics Processing Unit) cards. We developed a parallel version of the former algorithm using an NVIDIA GPGPU card (from now on GPU) with the unified architecture and the CUDA [28, 29] environment. To do this, a special version of basic linear algebra subprograms (BLAS) for GPUs, called CUBLAS [30] was used. Moreover, we developed a MATLAB toolkit for GPGPUS [31, 32] to use our implementation from MATLAB in such a way [33] that if the user has a graphic card, the performance of the implementation is improved. If the user does not have such a card, the algorithm can also be run using the machine CPU. Experimental results on a NVIDIA K20 (Kepler) latest (2013) [34] generation card was used. We compared the results with the code from MATLAB R2012b, using an INTEL QuadCore i7-3820 (3.6Ghz) CPU. All the results are shown in flops and seconds.

In our algorithm the most important computational cost it is due the number of square matrix products [35]. Using this, the computational cost of our problem in flops, can be approximated as the number of products times the cost of a product.

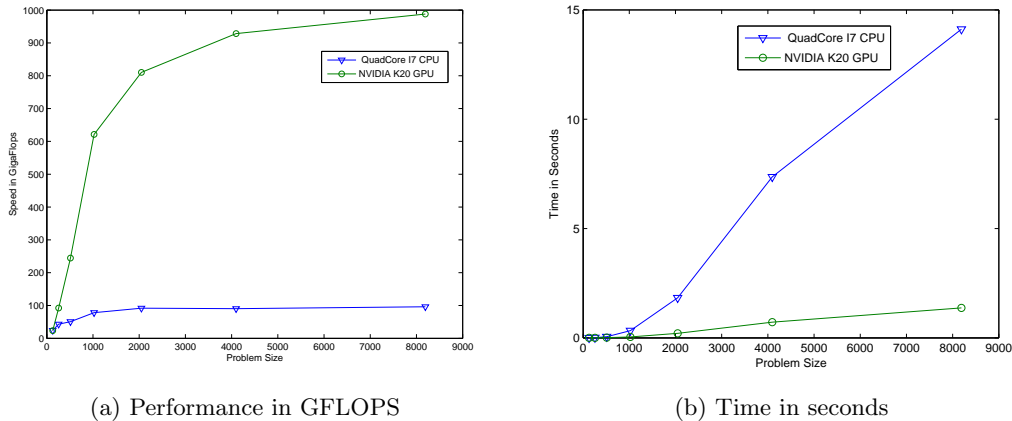


Figure 2: GPU (K20) vs CPU (i7 Quadcore) in GFLOPS (a) and seconds (b) when increasing the problem size.

In this test, we have used matrix dimensions equal to 2^n , $n = 7, 8, \dots, 13$, comparing the GPU vs the CPU performance in flops and seconds. All the matrices were generated randomly changing the matrix 1-norm, from 1 to 100 (in steps of 10). Each test was repeated 3 times.

From Table 1 and Figures 2a, we can see that GPU is much faster than CPU, inasmuch the GPU is a state of the art NVIDIA K20 while the CPU is an INTEL i7 CPU, not an INTEL XEON CPU. Anyway, our parallel implementation performance is good because it reaches almost the peak of the GPU card (1Tflop) [34]. Speedup refers to the number of times the parallel algorithm (using the GPU) is faster than the sequential algorithm (using the CPU).

6 Conclusions

An algorithm for computing the hyperbolic matrix cosine, based on a Hermite matrix polynomial expansion, and a new bound of the absolute error are presented. The numerical experiments show that the MATLAB implementation developed has lower execution times and higher accuracy than the MATLAB function `funm`. Also, we have developed a parallel algorithm, which use a state of the art GPGPU, an NVIDIA K20 card, to see if the implementation based on the above algorithm could efficiently compute the hyperbolic matrix cosine. The results have been good because in all cases, the parallel version works much faster than the serial version. Even, we almost get the theoretical maximum performance for the K20 when running our implementation (1 Teraflop/s). This can be interesting for solving large scale problems.

References

- [1] M. Sezgin. Magnetohydrodynamics flows in a rectangular duct. *International Journal for Numerical Methods in Fluids*, 7(7):697–718, 1987.
- [2] A. King and C. Chou. Mathematical modeling simulation and experimental testing of biochemical systems crash response. *Journal of Biomechanics*, 9:301–317, 1976.
- [3] L. Jódar, E. Navarro, and J.A. Martín. Exact and analytic-numerical solutions of strongly coupled mixed diffusion problems. *Proceedings of the Edinburgh Mathematical Society*, 43:269–293, 2000.
- [4] V. Soler, E. Defez, M. V. Ferrer, and J. Camacho. On exact series solution of strongly coupled mixed parabolic problems. *Abstract and Applied Analysis*, 2013.

- [5] D. Pozar. *Microwave Engineering*. Addison-Wesley, New York, 1991.
- [6] P. Das. *Optical Signal Processing*. Springer, New York, 1991.
- [7] L. Jódar, E. Navarro, A.E. Posso, and M.C. Casabán. Constructive solution of strongly coupled continuous hyperbolic mixed problems. *Applied Numerical Mathematics*, 47(3):477–492, 2003.
- [8] Nicholas J. Higham. *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2008.
- [9] J. Sastre, J. Ibáñez, P. Ruiz, and E. Defez. Efficient computation of the matrix cosine. *Applied Mathematics and Computation*, 219:7575–7585, 2013.
- [10] G. I. Hargreaves and N. J. Higham. Efficient algorithms for the matrix cosine and sine. *Numerical Algorithms*, 40:383–400, 2005.
- [11] Awad H. Al-Mohy and Nicholas J. Higham. A new scaling and squaring algorithm for the matrix exponential. *SIAM J. Matrix Anal. Appl.*, 31(3):970–989, 2009.
- [12] J. Sastre, Javier J. Ibáñez, E. Defez, and Pedro A. Ruiz. Accurate matrix exponential computation to solve coupled differential models in engineering. *Mathematical and Computer modelling*, 54:1835–1840, 2011.
- [13] E. Defez, J. Sastre, Javier J. Ibáñez, and Pedro A. Ruiz. Computing matrix functions solving coupled differential models. *Mathematical and Computer Modelling*, 50(5-6):831–839, 2009.
- [14] E. Defez, J. Sastre, Javier J. Ibáñez, and Pedro A. Ruiz. Computing matrix functions arising in engineering models with orthogonal matrix polynomials. *Mathematical and Computer Modelling*, 57(7-8):1738–1743, 2013.
- [15] Magnus Fontes, Michael Günther, and Nicole Marheineke, editors. *Progress in Industrial Mathematics at ECMI 2012, Series: Mathematics in Industry 19*. Springer-Verlag Berlin Heidelberg, 2014.
- [16] E. Defez, J. Sastre, Javier J. Ibáñez, J. Peinado, and M.M. Tung. A method to approximate the hyperbolic sine of a matrix. *International Journal of Complex Systems in Science*, 4(1):41–45, 2014.
- [17] J. C. Cortés López, Lucas Jódar, and R. J. Villanueva, editors. *Mathematical modelling Social Sciences and Engineering*. Nova Science Publishers, Inc., New York, 2014.
- [18] L. Jódar, L. Acedo, J. C. Cortés, and M. Ehrhardt, editors. *Modelling for engineering, & human behavior 2013*. Instituto Universitario de Matemática Multidisciplinar, UPV, Valencia, 2013.
- [19] Michael S. Paterson and Larry J. Stockmeyer. On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM J. Comput.*, 2(1):60–66, 1973.
- [20] E. Defez and L. Jódar. Some applications of Hermite matrix polynomials series expansions. *Journal of Computational and Applied Mathematics*, 99:105–117, 1998.
- [21] L. Jódar and R. Company. Hermite matrix polynomials and second order matrix differential equations. *Journal Approximation Theory Application*, 12(2):20–30, 1996.
- [22] E. Defez, Michael M. Tung, and Jorge Sastre. Improvement on the bound of hermite matrix polynomials. *Linear Algebra and its Applications*, 434:1910–1919, 2011.

- [23] G. I. Hargreaves and N. J. Higham. Efficient algorithms for the matrix cosine and sine. *Numerical Algorithms*, 40:383–400, 2005.
- [24] J. Sastre, Javier J. Ibáñez, E. Defez, and Pedro A. Ruiz. Efficient orthogonal matrix polynomial based method for computing matrix exponential. *Applied Mathematics and Computation*, 217:6451–6463, 2011.
- [25] Nicholas J. Higham. The Test Matrix Toolbox for MATLAB. Numerical Analysis Report No. 237, Manchester, England, December 1993.
- [26] Nicholas J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl.*, 26(4):1179–1193, 2005.
- [27] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Math. Programming*, 91:201–213, 2002.
- [28] NVIDIA. *NVIDIA CUDA compute unified device architecture*, 2009.
- [29] NVIDIA Corporation. *CUDA C Programming Guide*, 5.0 edition, October 2012.
- [30] NVIDIA. *CUDA. CUBLAS library*, 2013.
- [31] NVIDIA. Accelerating MATLAB with CUDA Using MEX Files, 2007.
- [32] MathWorks. Application guidelines. Accelerating matlab code using gpus, April 2010.
- [33] J. Peinado, J. Ibáñez, V. Hernández, and E. Arias. Speeding up solving of differential matrix Riccati equations using GPGPU computing and MATLAB. *Concurrency and Computation: Practice and experience*, Vol. 24 (2012) , Issue 12, 1334-1348, 2012.
- [34] NVIDIA. Tesla gpu high performance computing for servers. 2013.
- [35] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, 1996.